

4. 3D Hierarchies for Animation

Ronan Boulic and Olivier Renault
Computer Graphics Laboratory
Swiss Federal Institute of Technology
Lausanne, Switzerland

Geometric representation associated with computer-assisted modeling has proven to be very useful to visualize and understand the structure and behavior of a wide spectrum of entities (Foley et al. 1990). But if we examine the scope of the current 3D geometric models, we notice that they tend to embrace a static view of the information structure. In this chapter, we are particularly interested in stating the requirements for the modeling of complex 3D environments with mobile components. Such a modeling approach has to integrate a dynamic view of these components such that specific goals can be achieved in space and time. A key requirement is to provide a representation of relative motions and this can be addressed through the use of hierarchical modeling.

In the first section we review some geometric models and the PHIGS graphic standard which could be considered suitable for the modeling of 3D hierarchies. Then, we focus on the representation of mobility as it was first studied in robotics. The following sections are dedicated to an overview of the related technical requirements and to their application to a data hierarchy. The last section covers the general techniques of interaction and animation of 3D hierarchies.

4.1. Existing Hierarchical Geometric Models

Some geometric models are, by definition, 3D hierarchies. This is the case of the octree and CSG representations. We now present briefly these hierarchical approaches and then study their scope and their adequacy to the requirement of relative motion representation.

4.1.1. Octree

The octree representation is a subset of the cell decomposition technique, in which a solid is decomposed into arbitrary cells and is represented by each cell in the decomposition. Octree representation gives a 3D systemization to this cell decomposition.

Octrees are a 3D extension of the 2D quadtrees. Figure 4.1 shows the octree representation of a solid. The hierarchical tree results in a recursive decomposition of a cubic region into eight equally sized octants, which are cubic regions. If a cube is partially full, it is decomposed; if a cube is empty or completely full, it is not decomposed. A complex scene, made of several objects, can be decomposed in two ways:

- the complete scene is considered as a whole and is decomposed using one global frame,
- the objects are decomposed into their local frame and the complete scene is decomposed into the global frame using the bounding box of the objects; in this case there are two levels of decomposition.

One problem with the octree representation is that even small changes require recalculating the data system. If one object of the scene is in motion we see that, for the first case, all the decomposition must be built again and for the second case only the global decomposition using the bounding box must be redone. In both cases this updating of the representation of the scene is heavily time-consuming. Furthermore, octree representation doesn't integrate the relative positioning of objects. Octree representation may be suited for static scenes and for optimization of high CPU-consuming algorithms as ray-tracing or path planning.

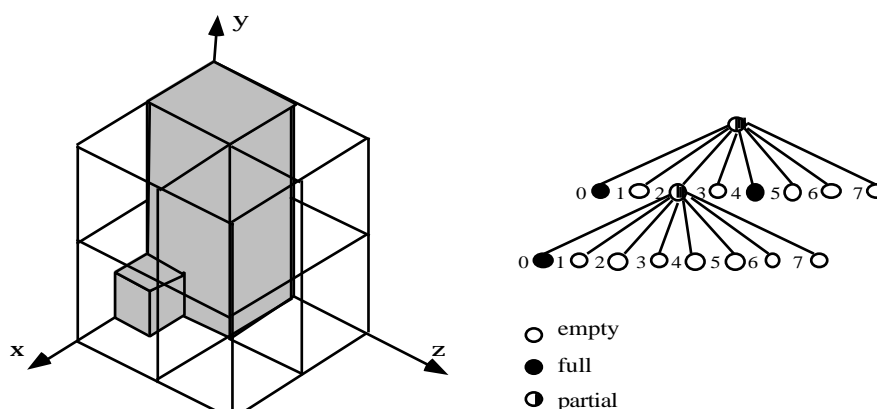


Figure 4.1. Octree representation of a 3D object

4.1.2. Constructive Solid Geometry (CSG)

CSG is a generalization of cell decomposition. Solids are created by assembling primitive volume objects along with three boolean operators: union, intersection and difference. Figure 4.2 gives an example of CSG representation.

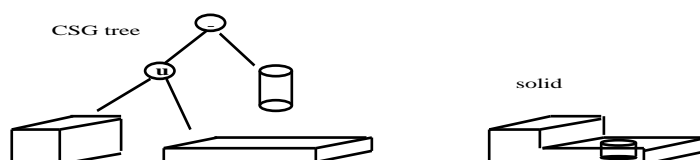


Figure 4.2. CSG decomposition of a 3D object

Objects are represented as binary trees, called CSG trees. Each terminal node is composed of a primitive object and a transformation (translation, rotation, scaling); each non-terminal node is either a boolean operator or a transformation which operates on the subnodes. This model, with its intuitive editing and its relatively compact storage of objects, is one of the most used representation for solid modeling in commercial packages.

CSG offers the feature of positioning objects of a complex scene relative to each other. The ability to edit the nodes allows independent motions of the components of a scene.

However, CSG presents two drawbacks for use in animation. First, its binary tree structure: scenes used in animation are easily describe with n-ary trees, as shown by Figure 4.3. The conversion of a n-ary tree into a binary tree leads to a less tractable representation. Finally, the main drawback is that CSG lacks of representation of mobility.

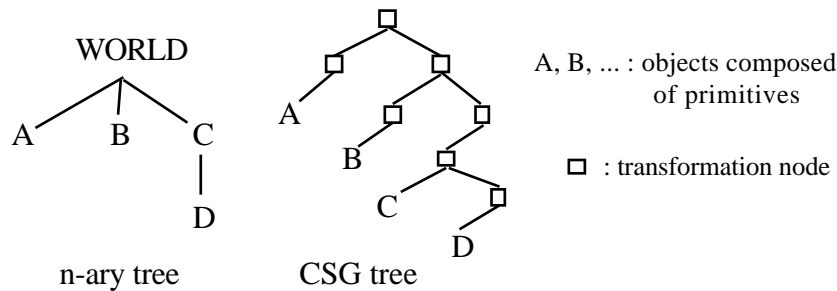


Figure 4.3. Conversion from n-ary tree to CSG tree

4.2. Representation of Mobility

This topic has been studied in the robotics field where control of articulated chains is the major goal. From a general point of view the relative mobility of two objects can be described with joints possessing from one to six *degrees of freedom (DOF)*. Figure 4.4 shows the usual joints with their number of degrees of freedom. Any of the multiple DOF joints can be decomposed into a series of one DOF joints.

Figure 4.4. Usual joints representing relative mobility of two objects

Therefore, it is only necessary to have some representations for one DOF joints so as to be able to represent a more complex one. Such a representation has been proposed (Denavit and Hartenberg 1955) which has become a common standard in this field. It employs only four parameters to describe the relative position of two axes of motion as shown on Figure 4.5.

Figure 4.5. Denavit–Hartenberg (DH) notation

The parameters can easily be derived from the typical chain structures of robotics which are a succession of parallel or perpendicular axes of motion. Although widely used in robotics, these parameters have some drawbacks as pointed out by Sims and Zeltzer (1988):

- By convention: the axis of motion of a link is the z axis of the preceding link in the chain, which can be confusing.
- This representation is directional in the sense it has to be covered from the base to the end of the chain to derive the global transformation matrices.
- It is not possible to utilize it for a tree structure or a closed loop.

An alternative principle of representation can be proposed to deal with this last point. It needs a set of seven parameters based on a decomposition in two parts:

- a constant part specifies the *shape* of the segment , with six parameters,
- a variable part specifies a one DOF motion (one parameter) along the new z axis.

This decomposition principle is used by Sims and Zeltzer (1988) which retains the position and orientation of the axes of motion as the shape and one more parameter for the degree of freedom. The fact that it is less directional than DH notation makes it very interesting for animation as further developed in Section 4.4.3 .

4.3. The PHIGS Standard

In this section we will present PHIGS (Programmer's Hierarchical Interactive Graphical System), an ISO standard for computer graphics programming. PHIGS allows to build, manipulate and visualize interactively 3D graphics data, providing the user with hierarchical data organization and editing capabilities.

4.3.1. Background

The design of PHIGS started in the early 80s. From the beginning PHIGS has been conceived as a standard. In an abstract way, a standard can be defined as a set of rules

permitting exchange of information through an interface, and so, independent development of the entities on both side of the interface. Thus, PHIGS has been designed as an interface between interactive applications that need complex graphics and graphics systems. It provides the application developers with a set of functionalities to manipulate and display their application models. Developing with PHIGS assures the programmers of the portability of their applications among different systems.

There are two worldwide official 3D graphic standards at the programmer-system level: GKS and PHIGS. GKS started as a 2D graphic standard and has been extended to 3D (GKS-3D) to meet the evolution of the graphic hardware. Basically PHIGS and GKS are constructed on a common set of graphic concepts and terminology, and PHIGS can be considered as a superset of GKS. The major differences between these two standards, concerning our topic, are (Shuey et al. 1986):

- GKS only offers a one-level, flat data organization,
- PHIGS allows more flexible, easier dynamic modifications of the graphic database.

4.3.2. Structure Hierarchy

In PHIGS, the basic organizational building blocks of graphic models are called structures. A structure consists of structure elements. A structure element can represent output primitives, attributes, viewing selections, modeling transformations or structure invocations. The graphic primitive elements are defined in their own coordinate system, called the modeling coordinate system. Modeling transformations convert the modeling coordinate space of the output primitives to world coordinate space. PHIGS allows multiple, cumulative modeling transformations. Attributes are used to define the appearance of the output primitives. A structure invocation element, called an EXECUTE-STRUCTURE element, is used to produce a directed acyclic graph (DAG).

To be displayed, a structure must be posted to the workstation. The display traverser goes over and executes elements of a posted structure one after the other; it applies the transformations, assigns attributes to the primitives and displays the output primitives. Attributes are modal: an attribute value specified by an attribute element is applied to all the primitives uncounted by the traverser until the next attribute element changes its value.

When an EXECUTE-STRUCTURE element is uncounted by the traverser the following actions occur:

- (1) the traversal of the current structure is suspended,
- (2) the state of the attributes values is saved,
- (3) the executed structure (and all the structures it executes) is completely traversed,
- (4) the attributes values are restored to the state saved before the executed structure was traversed,
- (5) traversal of the parent structure is resumed.

Traversal binding permits inheritance. An invoked structure inherits the defined attribute values and transformations of its parent structure. As a consequence of (4), a structure affects only those structures subordinated to it.

4.3.3. Limitation of PHIGS

Structure hierarchy in PHIGS allows the sharing of data. This sharing functionality can be useful to spare storage space but at the expense of independent management of the instances. As an example, we can construct a robot with an upper body invoking the same arm structure twice. Specification of modeling transformations before each invocation allows the left and right arm to be moved separately. If, in the arm structure, we add a call to an object structure so that the arm "holds" the object, both left and right arm will hold it. To avoid this we have to define two arm structures, one for the left and one for the right, each invoked only once. This is a classical tradeoff between instantiation of structure (multi-invocation of a same structure) and characterization (duplication of the structure) in structure hierarchies.

Although structure hierarchy allows inheritance of transformations and attributes, and so could be compared to procedure hierarchy, it lacks a general parameter-passing mechanism and a general flow-of-control construct. Furthermore, inheritance rules are very simple and do not support complex operations on solids as deformations. In fact, the PHIGS hierarchy is a data hierarchy extended with an elementary parameter passing mechanism but restricted from the dynamic editing point of view (Foley et al. 1990).

PHIGS carries some graphic limitations due to its early 80s conception. PHIGS+, an ongoing extension of PHIGS, tries to overcome these limitations by proposing new primitives (e.g. surface B-splines), better rendering (e.g. shading and different types of light), and limited control over the flow of execution of the structures.

4.4. Data Hierarchy: a Case Study

Unlike PHIGS structure hierarchies, data hierarchies can be managed dynamically. Although their flexibility has to be inscribed in the data themselves, for example using flags, this approach is still very convenient and, as such, widely used for scene representation.

4.4.1 Homogeneity vs Integration of Information

In the introduction, we stressed the fact that we were interested in modeling scenes which are basically complex, evolving environments. Such an entity has to integrate many different type of information, usually application dependant. How can we keep some homogeneity within the description of a 3D hierarchy with such an open stipulation ?

The first idea for matching the homogeneity requirement is to define a basic structure which is common to all the entities of the hierarchy. It retains the geometric, topological and display informations sufficient to set a frame in a 3D space within a hierarchy. We call it a `node_3D` (Figure 4.6).

In this way, one typed structure can be associated to a `node_3D` to represent additional functionalities. If a `node_3D` has no associated information it is said to be of NEUTRAL type ; it can be used as an intermediate frame to identify some specific location. The next paragraph describes how the typed data are partitioned into two families to further ensure the geometric integrity of the structure.

4.4.1.1. The Skeleton

As we cannot make any *a priori* statement on what the information to integrate is, none of the application-dependent information can be part of the internal structure of the hierarchy. Therefore we partition the typed data into two families.

- (1) A small set of types are used to construct the internal and mobile part of the hierarchy which is called from now on the ***skeleton*** (Figure 4.6). The idea is to provide some autonomous structures with respect to external information depending on the application.
- (2) All other information is considered as external to the structure and attached as ***terminal*** nodes to the skeleton. Figure 4.6 shows two such basic types.

Members of the skeleton family are the following.

- NEUTRAL:** retains all the informations to set an intermediate 3D frame.
- JOINT:** describes a one DOF joint either translational or rotational. As stated in section 4.2., the other usual joints can be constructed with a series of such node_3D.
- FREE:** it is possible to use six JOINT node_3D to completely define the position and orientation of a corps in space. The corresponding chain is well fitted to coordinated treatments as those provided in robotics. Yet, the orientation is under some kind of Euler sequence which always has some singularity. Another drawback is the fixed suite of orientation angles which may not be suited to user needs. That's why the FREE type has been introduced to specify locally some arbitrary geometric transformation. It is useful in the editing stage but also in animating some independent objects. For example, a human body can be built of JOINT for the bony skeleton while its motion with respect to some external support will be defined with a FREE node_3D.

4.4.1.2. General and Application-Dependent Terminal Types

Although terminal data are typically application-dependent, we can at least bring out two basic types to relate this approach to classic geometric modeling and to provide some graphic display.

- FIGURE:** This type is used to integrate informations about geometrical modeling of static structured "objects". A choice of such a model, or family of models, has to match with the necessities of the application.

For example in a robotics application: a CSG model allows the derivation of physical properties and a Boundary representation a fast display. A Design application can managed numerous representations as spline curves, surface patches etc.

We actually use a Boundary representation which retains a polygonal surface representation in a local frame. This representation is associated with various display and management procedures.

CAMERA: This is one major critical link between data and user because the user accesses to 3D information via a 2D projection . There must be some compensation to the loss of dimension by high interactivity and suitable working modes. Flexibility of the viewing and viewed points is essential to access to hidden information .

We only present here some of the basic operating lookat modes. The camera model retains the classic data of projection type and parameters, near and far planes, zooming coefficient, and a local transformation. This latter is used in conjunction with a lookat mode, three additional parameters and an entry point on a view node_3D. Four lookat modes are of primary interest.

Default mode: the local transformation is the identity matrix so the camera motion is the one of its father node_3D (any node_3D of the hierarchy).

Lookat_to_the view_node: the viewing direction of the camera is set to look at the view node, usually a reference up-vector is used to gear the twist angle along the viewing direction.

Lookat_from_view_node: the local transformation is set to look around from the location of the view node. Two coefficients are interpreted as two polar angles to orientate the direction of viewing.

Bubble_lookat_to view_node: this mode is derived from the virtual sphere principle: the camera is always pointing to the origin of the view node and moving around on a sphere (two parameters) of constant radius (one parameter)

Some extended modes of camera motion can be found in Turner et al. (1991).

Figure 4.7 (see color section) shows a layout of an application based on this decomposition principle. All the squares in the 2D viewport represent a node_3D and are used for selecting. Examples of color codes are: white for NEUTRAL, orange for JOINT, red for CAMERA.

Other types of terminal data can be developed to meet the qualifications of the application, for example: LIGHT for rendering, SOLID for dynamic computation, CONSTRAINT for motion control.

Figure 4.6. node_3D, skeleton and some typed data

4.4.2. Calculus Efficiency vs Memory Space

In the classical space-time tradeoff, we definitely favour the calculus efficiency against memory space. This will be justified in the following design choices for geometric information.

4.4.2.1 Homogeneous Coordinate System

First of all, we use a homogeneous coordinate system (Figure 4.8). Although this approach presents an elegant way of unifying coordinate transformations by simple concatenation of homogeneous matrices, it is also guided by the fact that more and more workstations provide some hardware management of such matrices. More advanced ones even provide a matrix stack to minimize the traversal cost of the application hierarchy.

Figure 4.8. Coordinate system transformation with homogeneous representation

4.4.2.2 Direct-Inverse Transformations (DIT)

The second major assumption related to geometric manipulation is the extensive use of coordinate transformations **in both direction**. That's why we introduce the DIT entity which retains both the **D**irect and **I**nverse Transformations between two coordinate systems. The convention is relative to the node_3d possessing the DIT (Figure 4.9):

- The Direct (dir) Transformation goes away from the proprietary node_3D.
- The Inverse (inv) Transformation comes to the proprietary node_3D.



Figure 4.9. DIT orientation convention with respect to the node_3D it belongs to

4.4.2.3. Local Transformations

We have agreed to the decomposition principle (cf Section 4.2) to represent the relative mobility of two objects. A constant initial DIT localizes each node_3D with respect to his father node. In addition, some types of node_3D (JOINT, FREE, CAMERA...) may hold a variable DIT. These are applied after the initial transformation (Figure 4.10).

4.4.2.4. Global Transformations

In addition to the local DIT, each node_3D hold a global DIT, also called reference DIT, with respect to a Reference node_3D. This latter is usually the root of the hierarchy (Figure 4.10).

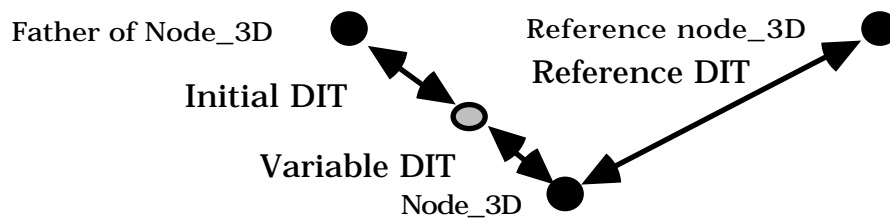


Figure 4.10. Basic local and global DIT maintained by a node_3D

The increased cost due to the updating of this information pays off as soon as we need to visualize the scene because it holds the equivalent concatenation of the traversal of the hierarchy. Then, if we wish to display the scene as viewed from a CAMERA node_3D, we just have to update the geometric pipeline matrix stack as follows:

- (1) Only once: multiply the top of the matrix stack with the inverse matrix of the camera reference DIT.
- (2) For each node_3D of the hierarchy:
 - (a) push the matrix stack and multiply it by the Direct matrix of the node_3D reference DIT,
 - (b) send the display commands expressed in its local frame,
 - (c) pop the matrix stack.

Many other algorithms can benefit from this information, in fact, as soon as any relative transformation has to be evaluated (Figure 4.11). For example, editing the lookat transformation for a camera, geometric reasoning, obstacle avoidance.

4.4.2.5. Memorization of Variable Information

As soon as the user wants to control realistically the evolution of a variable over time he needs to evaluate its current first and second derivatives. For this purpose, the JOINT and FREE types memorize the two previous values of their degrees of freedom. Some utilities can take profit of this memory buffer to retrieve and reconstruct up to the second previous state of the hierarchy.

Figure 4.11. Calculating the relative transformations between node_3D A and B

4.4.2.6. Transient Transformations

In addition to the preceding default local and global transformations we felt the need of one complementary optional feature to manage the transient geometric information. This feature may be very useful along various processes from edition to simulation.

In the editing process of working on some complex structure to obtain a specified attitude, the user is likely to appreciate an incremental approach, viewing both the current configuration while interactively editing a transient one. Once he is satisfied with the transient attitude he can validate it and it becomes the new current attitude.

In a simulation context, many algorithms work within a prediction–correction scheme. The current state of the system is maintained while the prediction and the correction are evaluated on the transient state. If it succeeds the correction is shifted to the current state. If it fails, a new transient prediction is evaluated, usually from the current state and a shorter timestep.

We have chosen to optionally retain a transient variable DIT for JOINT and FREE node_3D. In such a case it forces a transient global DIT to be retained for any influenced node_3D in the hierarchy. This transient global DIT is also evaluated with respect to the reference node_3D. Figure 4.12 (see color section) shows the edition of a leg posture: in magenta the current leg configuration and in red the transient left leg configuration. The user is interactively setting the transient and current left knee flexion.

Combined with the memorization of variable informations the transient information greatly improve the real-time efficiency of animation algorithms.

4.4.3. Topology and Animation

A hierarchical topology orientates the propagation of motion from the root to the terminal nodes. This property suits structures in which the base is fixed in the world frame, but is this always the case ? Let us first recall that a base, or root of motion, can be any node of the hierarchy except the WORLD node which is needed as an invariable reference. What happens if we want to represent some structure whose "behavior" induces a variable root of motion. For example where should be the root of motion for a human body ? And how can we change it if required by the animation ?

Most authors have stressed the ability to change the motion root of a hierarchy. For Kroyer (1986), the primary action devoted to the structure should be localized at the root, the secondary actions being tuned after the primary one. For example, in the context of a human

body structure, he chooses a BODY root localized at the base of the spine to define a walking motion. He justifies this by the fact this action clearly determines a forward motion of the body center of gravity which is close to the BODY node. Moreover he advises setting the root of the hierarchy at one hand if the human model has to hang at some bar. His approach of walking is not shared in Ridsdale et al. (1986) where the root of motion turns alternatively from the foot in contact with the floor to the next one. It has the clear advantage to prevent the supporting foot from going into the floor but conversely it is really difficult to control the speed of the body. That is why most of the walking studies refer to the first approach with some additional functionalities (cf Section 4.3.2).

Besides, authors are divided between those who prefer to redefine the topology of the hierarchy as Kroyer (1986), Cary et al. (1990) and Ridsdale et al. (1986) and those who rather redefine the traversal of the hierarchy, Van Baerle (1986) and Sims and Zeltzer (1988). This latter approach is more suitable in a context of dynamic location of the motion root and we actually subscribe to it.

4.4.3.1. Docking

Let us examine now how we can define a root of motion different from the topological root. This operation is called **docking** and the root of motion the **dock** node.

As the WORLD frame is invariant in space, we can see from Figure 4.12 that each child-hierarchy of the WORLD node_3D can be manipulated independently. This can be useful to organize complex scenes where each of these child-hierarchies can be assigned to a specialized functionality. For example for an animation set-up there can be some independent human structures, separate lights, cameras and decor configurations.

In this simple docking context, the traversal procedure which updates the global DIT checks if each WORLD child node_3D is in docking mode. Then it eventually deviates from the default downward chaining of transformations to an alternative upward and downward chaining beginning at the dock node_3D (Figure 4.13). The necessary information is kept with the WORLD data within a dynamic list of DOCK data. It comprises the root and dock entry points and the dock initial and local DIT with respect to the father node_3D of the root.(Figure 4.13).

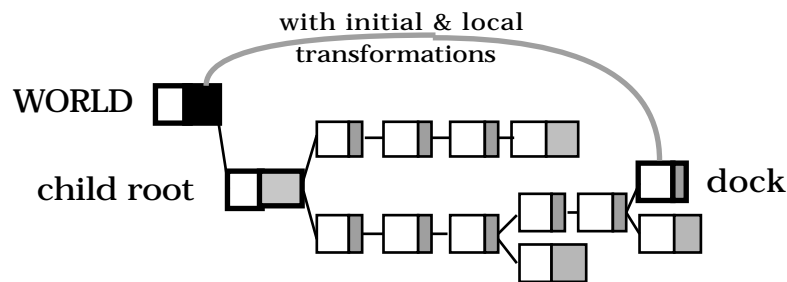


Figure 4.13. Information to retain for simple docking

A more general approach is to decentralize these DOCK data to permit local docking of a branch inside the hierarchy. In such a case, all node_3D should maintain their docking status either *root* / *dock* / *else* and an entry point to a shared DOCK structure.

4.4.3.2. Sharing

Considering the design choices we have made for the geometrical and topological informations, sharing can easily be performed on terminal data types.

The terminal node_3D holds an horizontal node_3d list of all the terminal node_3d sharing the same typed data (Figure 4.14). With this sharing scheme, the instances can still be characterized geometrically and visualized independently.

Proposing a sharing scheme for sub-hierarchies requires some additional features and management, particularly for the global transformations. How can we describe global situation of multiple instances? Duplication of these transformations would be hard to maintain at the level of each shared node_3D with respect to the corresponding father nodes of the sub-hierarchy root. A better solution is to reference all the global transformations of a shared sub-hierarchy to its root node_3D ; thus they become sub-global transformations. The final WORLD transformation is then obtained by concatenation of the sub-global transformations.

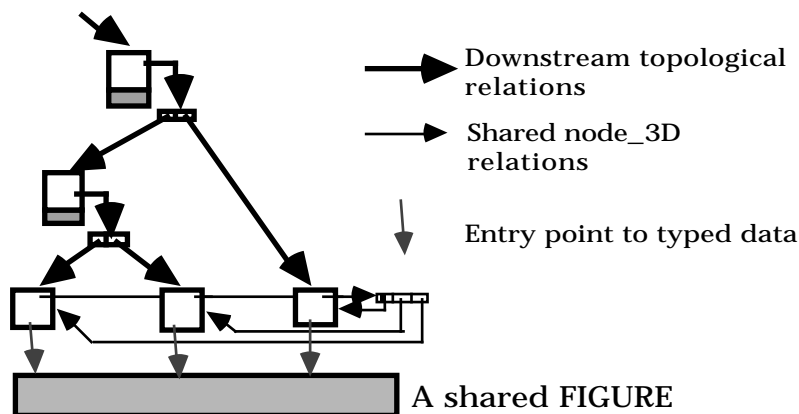


Figure 4.14. node_3D sharing scheme of terminal typed data

4.4.3.3. Editing

It is sometimes necessary to modify the topology to transfer the motion control of a branch from its current father to a new one ; this is the typical case of "pick and place" tasks in robotics. Transfer is a rather straightforward operation unless the desired new father node_3D is a terminal one. In such a case a NEUTRAL node_3D can be created with the same location and the same father and the transferred branch is attached to it.

4.5. Animation Techniques

4.5.1. Control Spaces

Let us give some definitions before going further:

- The whole set of local transformations corresponds to the **configuration space** of the structure. Its dimension, noted \mathbf{n} , is the sum of all the degrees of freedom.
- The global transformation corresponds to the **situation space** of a node_3D. It is six dimensional for full specification of the position and orientation of a node 3D.

In addition to these spaces and their derivative spaces, other control spaces can be used as force-torque subject to having the necessary inertial data (they could be in a SOLID type).

4.5.2. Geometric Control

By construction of the hierarchy, specification of configuration is the default control mode. Local transformations are defined and global transformations are updated depending on the traversal direction (Boulic and Renault 1991).

Specification of situation is the inverse problem but it may have zero, one, a finite number or an infinite number of solution(s). In robotics, most of the commercial robots match a few well known chain structures for which there exist specialized algorithms to deal with this issue. When the robot has as many degrees of freedom as cartesian dimensions to control, there is always a finite number of solutions. The final choice is made with some additional stipulation as elbow *high* or *low* etc. A higher level stipulation is to evaluate the feasibility of some trajectory beginning at the desired situation (Borrel 1986).

We need an alternative method to cover the general context with any kind of chain structures. This method has also to deal with **redundant** systems with more degrees of freedom than cartesian dimensions to control, leading to an infinite number of solutions. A local approach based on a linearization of the system at the current state can be used to obtain a configuration solution close to the current one. This tool is widely used in robotics and animation (Cary et al. 1990) and is called inverse kinematics.

4.5.3. Kinematic Control

4.5.3.1. Keyframe

Keyframe methods control the evolution of some parameters over time with only a reduced list of key_values of the parameters set at key_time. This defines some *control points*. Traditionally, key_times are expressed by a frame number of an animation film but this is no more a requirement. There exist a few interpolation or approximation functions based on groups of four control points with some additional qualitative parameters such as tension, and bias (Magenat-Thalmann and Thalmann 1990).

4.5.3.2. Functional Models

The previous parametric animation method becomes limited for the representation of complex coordinated motions. For this reason many authors (Zeltzer 1982; Girard and Maciejewski 1985; Sims and Zeltzer 1988; Boulic et al. 1990b; Maiocchi 1991) stress the need for functional model providing some higher level input parameters from which low level parameters values are automatically derived. The idea is to build an animation system with a library of such models and composition operators. To date most of the researches focused on human or animal locomotion. The recent models usually combine configuration and situation control with inverse kinematics (cf next section).

Another related approach, although less fundamental to the understanding of the motion, is data acquisition with rotoscopy (Maiocchi and Pernici 1990) with further editing or composition. It usually lacks a real functional level input (as speed for walking motion, for example).

4.5.4. Inverse Kinematics

Considering the problem of modifying the situation of a node_3D with an opened chain, the general discrete form of the solution provided by inverse kinematics is

$$q = J^+ x + (I - J^+ J) z \quad (4.1)$$

- q is the solution of the equation in the configuration variation space.
- x describes the *main task* to realize in term of a variation of the situation of the controlled node_3D. (Figure 4.15 a,b,d: a translation in the cartesian plane).
- I is the identity matrix of the configuration variation space (n x n)
- J is the Jacobian matrix associated to the controlled system
- J⁺ is the unique pseudo-inverse of J providing the minimum norm solution which realizes the *main task* (Figure 4.15 a,b).
- (I - J⁺J) is a projection operator on the *kernel* of the application describing the main task
- z describes a *secondary task* in the space of variation of configuration; this task is partially realized via the projection on the *kernel*. In other words, the second part of equation does not modify the achievement of the main task for any value of z (Figure 4.15 c has a null vector as main task to materialize the kernel space). Usually z is calculated so as to minimize a cost function.

If the dimension of the main task is m, then the kernel is (n-m) dimensional in the configuration variation space. This information is fundamental to evaluate the potential of the secondary task.

Some secondary tasks that have been used in robotics are avoiding joints limits (Borrel 1986), and optimizing maneuverability. In animation it is applied in the same way and as an interactive tool to tune a configuration while respecting a fixed situation for the controlled node_3D. For example, Girard and Maciejewski (1985) fix the pelvis and foot location and fine tunes the intermediate configuration of the leg via the secondary task. More recently, Boulic et al. (1990a) propose a methodology of correction of predefined motions. The idea is to retain the most of an initial motion by direct kinematics while applying some simple

constraints to be fulfilled by inverse kinematics. This methodology, currently under investigation, is interested in providing a mixed kinematic control allowing both direct and inverse transitions.

4.4. Dynamic Control

There has been considerable interest in the introduction of dynamic control for complex structures (Isaacs and Cohen 1987; Wilhems 1990). This kind of control of hierarchy is well treated as long as there are no closed loops or complex interactions (as in walking between feet and floor). In such context, authors have used a mixed kinematic and dynamic control scheme (Girard 1987; Sims and Zeltzer 1988; Maiocchi 1991).

Furthermore, such a force-torque control space shall be suitable to interactive specification only when there exist appropriate devices to provide input and output of this nature.

Figure 4.15. (a) and (b) main task only, (c) secondary task with null vector main task, (d) same main task as (b) with arbitrary secondary task

4.5. Limitations of Hierarchical Structure

As pointed out in the previous section, the representation and management of closed loops is close to going beyond the scope of hierarchical representation. That is why most mechanical simulation systems represent solid systems by a connected graph with solid objects as nodes and joints as arcs (Hégon 1988; Cremer 1989).

In fact, numerous fields of geometric modeling require graph structures to represent networks or complex layouts such as electrical circuits, chemical plants and molecules.

4.6. Conclusion

We have reviewed the basic requirements and the scope of modeling complex 3D environments with hierarchies. Although designed for such purpose the PHIGS standard doesn't address all the key issues as flexibility, calculus efficiency, integration of new functionalities, such as cameras and rendering, and motion root modification. In a second part, we have presented a framework for data hierarchy that tries to embody these qualifications. The last section terminates with a classification of the available motion techniques.

Acknowledgement

We wish to thank Russell Turner for his help in reviewing this chapter.

References

- Borrel P (1986) *Contribution a la modélisation géométrique des robots manipulateurs. Application a la conception assistée par ordinateur*, Thèse d'Etat, Université des Sciences et Techniques du Languedoc, Montpellier.
- Boulic R, Magnenat-Thalmann N, Thalmann D (1990) A New Methodology for the Correction of Predefined Motions", *EUROGRAPHICS Workshop on Animation and Simulation*, EPFL Lausanne.
- Boulic R, Magnenat-Thalmann N, Thalmann D (1990b) A Global Human Walking Model with real time Kinematic Personification, *The Visual Computer*, Vol 6, No6.
- Boulic R, Renault O (1991) Modélisation d'objets hiérarchiques tridimensionnels, Notes du XII Cours Postgrade en Informatique Technique, EPFL, Lausanne.
- Cary BP, Zhao J, Badler NI (1990) *Interactive Real-Time Articulated Figure Manipulation using Multiple Kinematic Constraints*, ACM.
- Cremer JF (1989) The architecture of NEWTON, a general purpose Dynamics simulator, *Proc. IEEE Conf. on Robotics and Automation*.
- Denavit J, Hartenberg RS (1955) A Kinematic Notation for Lower Pair Mechanisms Based on Matrices, *J. Applied Mechanics*, Vol 22.
- Foley J, Van Dam A, Feiner S, Hughes J (1990) *Computer Graphics Principles and Practice*, Second edition, Addison-Wesley, Reading, MA.
- Girard M, Maciejewski AA (1985) Computational Modeling for the Computer Animation of Legged Figures, *Proc. SIGGRAPH '85, Computer Graphics*, Vol.19, No3, San Francisco, CA.
- Girard M (1987) Interactive Design of 3D Computer-Animated Legged Animal Motion, *IEEE Computer Graphics and Applications*, Vol.7, No6.
- Hégon G (1988) *Animation de systèmes de corps rigides*, Notes de cours de Computer Animation International 88, Genève.
- Isaacs PM, Cohen F (1987) Controlling dynamic simulation with kinematic constraints, behavior functions and Inverse Dynamics, *Proc. SIGGRAPH '87, Computer Graphics*, Vol.21, No4.
- Kroyer B (1986) Animating with a Hierarchy, *SIGGRAPH '86 Course notes of the Seminar on Advanced Computer Animation*.
- Magnenat-Thalmann N, Thalmann D (1990) *Computer Animation: Theory and Practice*, second edition, Springer Verlag, Tokyo.
- Maiocchi R, Pernici B (1990) Directing an Animated Scene with Autonomous Actors, *The Visual Computer*, Vol. 6, No6, Springer Verlag, Heidelberg.
- Maiocchi R (1991) A knowledge-based approach to the synthesis of human motion, in: Kuniti (ed.) *Modeling in Computer Graphics*, Springer Verlag, Tokyo.
- Ridsdale G, Hewitt S, Calvert TW (1986) The Interactive Specification of Human Animation, *Proc. Graphics Interface and Vision Interface '86*.
- Sims K, Zeltzer D (1988) A Figure Editor and Gait Controller for Task Level Animation, *SIGGRAPH '88 Course notes on Synthetic Actors*.
- Shuey D, Bailey D, Morrissey TP (1986) PHIGS: A Standard, Dynamic, Interactive Graphics Interface", *IEEE Computer Graphics and Application*, Vol.6, No8.

- Turner R, Balaguer F, Gobetti E, Thalmann D (1991) Physically-based interactive camera motion using 3D input devices, *Proc of CGI '91*.
- Van Baerle S (1986) Character Animation: Combining Computer Graphics and Traditional Animation, *SIGGRAPH '86 Course notes*.
- Wilhelms J (1990) Dynamic Experiences, in: Badler N, Barsky B, Zeltzer D (editors) *Making Them Move*, Morgan Kaufmann Publishers, pp.265-279.
- Zeltzer D (1982) Motor Control Techniques for Figures Animation, *IEEE Computer Graphics and Applications*, Vol.2, No9, pp.53-59